AD-A285 720

NAVAL POSTGRADUATE SCHOOL
Monterey, California

94-32943

THESIS

DTIC QUALITY INSPECTED 2

OBJECT-ORIENTED METHODOLOGY FOR MARINE CORPS SOFTWARE DEVELOPMENT

by

Robert F. Padilla, Jr.

September, 1994

Thesis Advisor: C. Thomas Wu

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: OBJECT-ORIENTED METHODOLOGY FOR MARINE CORPS SOFTWARE DEVELOPMENT (U)	5. FUNDING NUMBERS
6. AUTHOR(S) Robert F. Padilla, Jr.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT *(maximum 200 words)*. This thesis answers three questions. What is object-oriented development methodology and why is it good for the Marine Corps? How is object-oriented methodology different from what the Marine Corps is doing now? What should the Marine Corps do and when should they do it?

To explore these issues, this thesis designed a typical Marine Corps application (a COmpany Personnel System (COPS)) using both Systems Development Methodology (SDM) and Object Modeling Technique (OMT). These methodologies are compared in terms of ease of maintenance, understandability, extendibility, inheritance, and database integration.

It is good for the Marine Corps because it helps developers and customers express abstract concepts clearly. OMT and SDM differ in their approach to system organization: OMT around real-world objects, while SDM around functionality. The Marine Corps should immediately change its paradigm from SDM to OMT. SDM's Functional Requirements Definition, General Design Specification, and Detailed Design Specification will have to be replaced with OMT's Analysis, System Design, and Object Design respectively.

14. SUBJECT TERMS Object-Oriented Methodology, System Development Methodology (SDM), Object Modeling Technique (OMT)			15. NUMBER OF PAGES 64
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

Approved for public release; distribution is unlimited.

Object-Oriented Methodology for Marine Corps Software Development
by

Robert F. Padilla, Jr.
Captain, United States Marine Corps
B.S.B.A., Ohio State University, 1987
M.B.A., United States International University, 1992

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: _____
Robert F. Padilla, Jr.

Approved by: _____
C. Thomas Wu, Thesis Advisor

_____
Lou Stevens, Second Reader

_____
Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

This thesis answers three questions: What is object-oriented development methodology and why is it good for the Marine Corps? How is object-oriented methodology different from what the Marine Corps is doing now? What should the Marine Corps do and when should they do it?

To explore these issues, this thesis designed a typical Marine Corps application (a COmpany Personnel System (COPS)) using both Systems Development Methodology (SDM) and Object Modeling Technique (OMT). These methodologies are compared in terms of ease of maintenance, understandability, extendibility, inheritance, and database integration.

It is good for the Marine Corps because it helps developers and customers express abstract concepts clearly. OMT and SDM differ in their approach to system organization: OMT around real-world objects, while SDM around functionality. The Marine Corps should immediately change its paradigm from SDM to OMT. SDM's Functional Requirements Definition, General Design Specification, and Detailed Design Specification will have to be replaced with OMT's Analysis, System Design, and Object Design respectively.

Accesion For

NTIS

DTIC

By

Dist

A-1

TABLE OF CONTENTS

I. INTRODUCTION

The United States Marine Corps is currently evaluating object-oriented methodology for several future software development projects because complex software developed using that methodology appears to be faster to develop and easier to maintain. This thesis will explore the required changes in current procedures and standards in the area of software engineering techniques and will provide Marine Corps officials with an objective look at changes in software engineering procedures that are inherent in a paradigm shift.

A. BACKGROUND

The advantages of object-oriented methodology have been widely discussed in the technical press and there appears to be a widespread move in that direction for commercial applications and software development To understand the impact of this new methodology on Marine Corps software development processes requires an assessment of object-oriented capabilities, limitations, and constraints Additionally, it requires an understanding of what changes must be made to current Marine Corps software development methodologies

The Marine Corps uses two sets of standards to guide the development of software, one standard is for unique "tactical" systems, the other standard is for "non-tactical" systems This thesis will explore only the software development methodologies for

non-tactical systems. These methodologies are similar to those used by private industry, with slight modifications specific to military use. In addition, this thesis will explore only software development for PC's and network servers. The development methodology for Marine Corps mainframes is the same What differs is the level of detail required by the standards and procedures. The Marine Corps defines three categories of systems; small, medium, and large. These categories are defined by the amount of money spent on system development and implementation. Small systems are limited to less than $1 million, medium systems run between $1 million to $5 million, and large systems cost over $5 million. The larger the system, the larger the documentation requirements Systems developed for PC's and network servers tend to fall into the category of small systems. By limiting the scope to small systems, this thesis will be able to focus on the details of development methodology, instead of getting caught up in unnecessary documentation requirements. Furthermore, this thesis looks at the use of object-oriented software methodology as it applies to a proposed or existing Marine Corps application. The intent is to show the generic changes that are required and provide Marine Corps officials with the general idea behind, and impact of, object-oriented methodologies.

B. USE OF CURRENT METHODOLOGY

Currently, the Marine Corps uses a design methodology based on traditional structured analysis and design principles (these will be defined in Chapter II). However, based on practical experience, it can be said that these standards and procedures are not always adhered to by system developers Due to time constraints and other factors, most

small systems are not designed properly, that is, following the standards and procedures It seems that once a requirement has been defined, the programmers start coding the system, without taking the initial time up front to properly design and document the proposed system. A fundamental shift in attitude towards proper system development must be embraced by Marine Corps system developers. Once this has been accomplished, the next step is to adopt a methodology that organizes a system around real-world objects, instead of around procedures. The object-oriented approach to systems analysis and design is presented in this thesis.

In order to define what the object-oriented approach to systems analysis and design is, and what the impact of such an approach is, the following list of questions will be addressed and answered in this thesis The questions have been broken down into three categories

A What is it and why is it good for the Marine Corps?

* What is object-oriented software design?
* What are the benefits of systems developed with object-oriented methodologies?

B How is it different than what we are doing?

* What are the current methodologies used for software development?.
* What changes are required to current development methodologies in order to develop software based on object-oriented methodologies?

C. What should we do and when should we do it?

* When is an object-oriented approach beneficial?
* Is object-oriented methodology applicable to Marine Corps non-tactical software development activities?

Categories A and B serve as background subjects, while category C will show the benefits and applicability of a object-oriented approach to Marine Corps applications

C. METHODOLOGY

In order to provide for background, as well as applicability, this thesis will follow a number of certain steps. This thesis will report the results of a five step process, defined below.

Step one: Understand Current Procedures. This step will explore current Marine Corps software development procedures and standards. This will be done in order to gain an understanding of the requirements that drive current software development. This is essential in order to show where changes must be made if object-oriented methodologies are to be adopted.

Step two: Understand Object-Oriented Procedures. This step will explore object-oriented methodologies, specifically in the area of software engineering. This is essential in order to compare and contrast current methodologies with object-oriented methodologies. The Object Modeling Technique (OMT) will be used as the object-oriented methodology.

Step three: Identify Elements of Applicability. This step will determine if (and when) object-oriented methodologies are applicable to Marine Corps non-tactical software development. This will focus on a proposed or existing system.

Step four: Compare the Methodologies. This step will compare and contrast the two methodologies. The strengths and limitations of each will be addressed. Focus will be given to the area of "where changes are to be made" if object-oriented methodologies are to be applied and practiced.

Step five: Conclusions and Recommendations. Finally, conclusions and recommendations will be made about the use of object-oriented procedures and techniques for Marine Corps software development activities.

D. ORGANIZATION OF THESIS

This thesis is organized as follows. The next chapter, Current Marine Corps Software Development Standards and Procedures, describes current Marine Corps standards and procedures with respect to system development and analysis.

The third chapter, Object-Oriented Software Design, defines an object-oriented approach to system analysis and design. Specifically, the Object Modeling Technique (OMT) is presented.

The fourth chapter, Applicability of Object-Oriented Methodology to Marine Corps Software Development, will look at an existing or proposed system (developed using current methodologies), and develop a portion of that system using OMT.

The fifth chapter, Comparison Between Current and Object-Oriented Methodologies, will compare and contrast the two methodologies.

The sixth chapter, Conclusions and Recommendations, will make specific recommendations to the Marine Corps about object-oriented methodologies.

II. CURRENT MARINE CORPS SOFTWARE DEVELOPMENT STANDARDS AND PROCEDURES

Current Marine Corps standards and procedures are based on the Department of Defense (DOD) and Department of the Navy's (DON) "Life Cycle Management" (LCM). "LCM is a process of administering an information system (IS) over its entire life with emphasis on strengthening early decisions which influence IS costs and utility" [Ref. 1]. The Marine Corps has used the LCM as a guideline for it's own version of the LCM. Within this version lies the methodology that governs all current Marine Corps software development activities; the System Development Methodology (SDM).

A. SYSTEM DEVELOPMENT METHODOLOGY

1. Background

The SDM is the formal specification for building a system. It defines the activities necessary to build a system, the interface between those activities, and control of the products created as a result of those activities [Ref. 2:p. 1-3]. The intent of the SDM is to provide a methodology based on existing government publications, but enhanced to accommodate the technologies and constraints specific to the development of new systems. SDM follows the guidelines set forth in Marine Corps Order P5231.1, "Life Cycle Management for Information Systems Projects (LCM-IS)," which provides overall policy on system development [Ref. 2:p. 1-4].

The SDM is based on a traditional software development activity, known as the software life-cycle. This life-cycle approach involves the following activities: requirements analysis, system specification, system design, detailed design, coding, integration, operation and maintenance. The activities that pertain to software engineering aspects within the SDM will be defined later.

2. Phases

The phases of the SDM are similar to those of the LCM, but with a couple of differences. There is an additional division within the Definition and Design phase and the System Development phase. These sub-divisions are necessary in order to incorporate the traditional formal structured design approach to software development. Figure 1, "System Development Phases," shows a comparison of the DOD/DON LCM with the Marine Corps SDM.

DOD & DON LCM

Mission Anal/ Project Init	Concept Development	Definition and Design		System Development			Deployment and Operations
		Definition	Design	Developmet and Integration	Test	Eval.	

MARINE CORPS SDM

Mission Anal/ Project Init.	Concept Development	Definition and Design			System Development				Deployment and Operations
		General		Detailed Design	Code	Development and Integration	Test	Eval.	
		Def.	Des.						

Figure 1 System Development Phases

The major emphasis in the SDM is placed on the actual development activities, which occur in the Concept Development through System Development Phases. This is where the majority of the software design and development work is done.

3. Documentation Requirements

The SDM requires that several documents be prepared and delivered to management. These documents are due at several different stages throughout the SDM process. This thesis will only address those documents that are pertinent to the software development process; namely the "Functional Requirements Definition" (FRD), "General Design Specification" (GDS), and the "Detailed Design Specification" (DDS). These documents are required in the Definition and Design phase of the SDM. Figure 2, "Deliverables of SDM", displays the document deliverables of the SDM.

Mission Analysis	Concept Development	Definition and Design		Development	Deployment and Operation
		General Design	Detail Design		
	Economic Analysis	Functional Requirements Definition and General Design Specification	Detailed Design Specification	Users Manual Computer Operations Manual	

Figure 2 Deliverables of SDM

There are more documents required under the LCM, but these documents do not pertain to the software engineering aspects of development. The documents that pertain to software engineering aspects will be described below.

a. Functional Requirements Definition

This standard provides the guidelines to produce a Functional Requirements Definition . The definition developed will detail the user's requirements for new, changed, or enhanced applications necessary to meet the system's objectives [Ref. 2:p. 2-10]. When completed, the functional requirements definition will be provided to project management for review and approval.

The FRD is the equivalent to the traditional life-cycle "System Specification" phase. It involves eliciting from a customer the behavioral characteristics and properties of the software system that is to be developed. They must be specified in an accurate, complete, unambiguous and non-contradictory way [Ref. 3]. The purpose of the system specification is to determine a customer's needs in sufficient detail to plan the construction of a software system meeting those needs [Ref. 4]. The end product of this activity will be a software specification document, which includes functional

requirements. A functional requirement is a statement of what a software system is to do; the functions it is to perform.

b. General Design Specification

This standard provides the guidelines to produce General Design Specifications. The specifications developed will detail the system environment for new, changed, or enhanced applications necessary to meet the system's objectives [Ref. 2:p. 2-10]. When completed, the GDS will be provided to management for approval.

The GDS is the equivalent to the traditional life cycle "System Design " phase. It involves defining the architecture of a system which satisfies the customer requirements expressed in the specification [Ref. 3:p. 4]. This is the process of design. This design process partitions the software into functionally related groupings, known as modules. These modules will consist of constants, variables, types and program units (functions and procedures) which provide resources to carry out a series of related tasks [Ref. 3:p. 4]. The result of the system design phase is a system architecture which defines the relationships between individual program units in a proposed system. The next step in the process is to develop a detailed design.

c. Detailed Design Specification

This standard provides the guidelines to produce a Detailed Design Specification. Basically, it develops the technical solution to the customers needs [Ref. 2:p. 2-9]. When completed, the DDS will be provided to management for approval.

The DDS is the equivalent of the traditional life-cycle "Detailed Design" phase. It is the process of transforming a system design into a form in which it can be given to a programmer and implemented. It involves filling in the details of the system design [Ref. 3:p. 5].

4. Structured Analysis and Design

The phases contained in the SDM are based on phases in the traditional software life-cycle process. Since the life-cycle process is based on structured analysis/design principles, the SDM follows these principles as well. The document deliverables of the SDM can be tied directly to the structured analysis/design approach to software development, as displayed in Figure 3, "Structured Systems Development Activities."

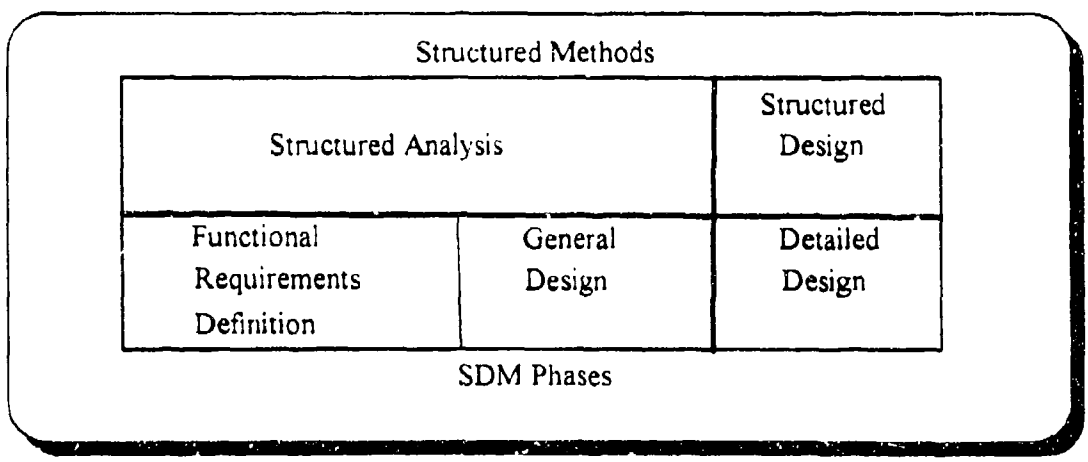

Figure 3 Structured Systems Development Activities

The next chapter in this thesis will address the Object-oriented approach to software development.

III. OBJECT-ORIENTED SOFTWARE DESIGN

The intent of this chapter is to present the reader with an overview of object-oriented software analysis and design techniques Object-oriented design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity [Ref. 5] This chapter will address the "most common" characteristics required by an obj. ct-oriented approach Once this has been done, a specific object-oriented software design methodology, Object Modeling Technique (OMT), will be presented and discussed

A. CHARACTERISTICS OF AN OBJECT-ORIENTED APPROACH

There is some dispute about what exactly characterizes an object-oriented approach Generally speaking, however, there are four widely accepted characteristics identity, classification, polymorphism, and inheritance [Ref 5:p 1] In addition to these four, there are two additional characteristics that need to be mentioned; abstraction and encapsulation

1. Identity

Identity means that data are organized into discrete, distinguishable entities called objects. Each object has its own inherent identity Two objects are distinct even if all their attribute values are identical.

In the real world an object simply exists, but within the context of a programming language, each object has a unique handle by which it can be referenced [Ref. 5 p. 2] Object references are uniform and independent of the contents of the objects, permitting mixed collections of objects to be created.

2. Classification

Classification means that objects with the same attributes (data structures) and operations (behavior) are grouped into a single class. A class is an abstraction that describes properties important to an application. Any choice of classes is arbitrary and is application specific [Ref. 5:p. 2]

3. Polymorphism

Polymorphism means that the same operation may behave differently on different classes [Ref. 5:p. 2]. What this means is that the result of the operation (method) will vary between classes. A specific implementation of an operation by a certain class is called a method. The user of an operation need not be aware of how many methods exist to implement a given operation. New classes can be added without changing existing code, provided methods are provided for each applicable operation on the new classes [Ref. 5:p. 3].

4. Inheritance

Inheritance is the sharing of attributes and methods among classes based on a hierarchical relationship [Ref. 5:p. 3]. This is known as the inheritance mechanism, with which a new class may be declared as an extension or restriction of a previously defined

class [Ref 6] This leads us to the notion of superclass and subclass Each subclass inherits all of the properties of its superclass and adds its own unique properties

The ability to factor out common properties of classes into a common superclass, and to inherit the properties from the superclass, greatly reduce repetition within design and program code [Ref 5 p 3]

5. Abstraction

Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring those aspects that are not important [Ref 5 p 16]. In system development, this means focusing on what an object is and does, before deciding on how it should be implemented By using abstraction, the designer maintains more flexibility by avoiding premature commitments to implementation details Proper use of abstraction allows for the same model to be used for analysis, high-level design, program structure, etc

6. Encapsulation

Encapsulation, also known as information hiding, consists of separating the external aspects of an object from the internal implementation details of the object [Ref 5 p. 7]. Encapsulation prevents a program from becoming so interdependent that a small change in one area will cause massive changes throughout

B. OBJECT MODELING TECHNIQUE

A methodology for software design is usually presented as a series of steps, with specific techniques and notations associated with each step. The OMT methodology

supports the entire software life cycle. The complete software life cycle spans from initial problem formulation, through analysis, design , implementation, and testing. This thesis will focus on OMT as it applies to analysis and design.

The OMT methodology consists of several phases These phases are Analysis, System Design, and Ooject Design

1. Analysis Phase

Analysis is concerned with devising a precise, accurate, understandable, and correct model of the real world The purpose of object-oriented analysis is to model the real world system so that it can be easily understood [Ref 5 p 148] It clarifies the requirements, provides a basis for agreement between the software requester and the software developer, and becomes the framework for later design and implementation

Analysis begins with a problem statement This is generated by clients and developers The real world system described by the problem statement is abstracted into a model This model addresses three aspects of objects static structure, sequencing of interactions, and data transformations Static structures are displayed in the Object Model, sequencing of operations in the Dynamic Model, and data transformations in the Functional model Figure 4 displays an overview of the Analysis process [Ref 5 p 149]

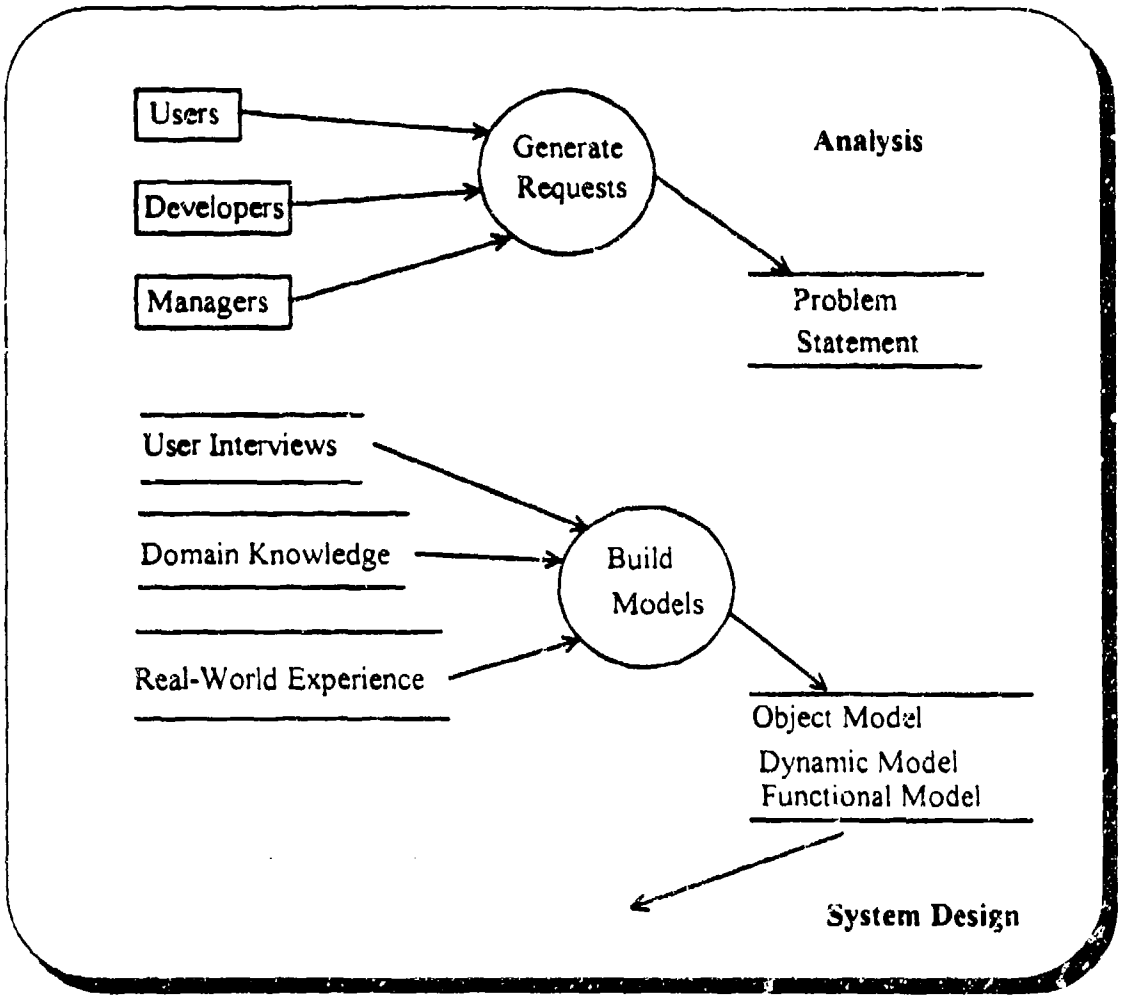

Figure 4 Overview of Analysis Process

Analysis is not a mechanical process. Large models are built up iteratively. First a subset of the model is constructed, then modified, until the complete problem is understood. The object, dynamic, and functional models will be described next.

a. Object Model

The object model describes the static structure of objects in a system. Their identity, relationship to other objects, attributes, and operations are described [Ref. 5:p.

17]. The object model is the most important of the three models. Building a system around objects rather than functionality is emphasized. The goal in constructing an object model is to capture those concepts from the real-world that are important to an application. The object model is represented graphically with object diagrams containing object classes. These classes are arranged into hierarchies sharing common structure and behavior and are associated with other classes. The steps necessary to build an Object Model are as follows [Ref. 5:p. 152]:

* Identify object classes.
* Prepare a data dictionary.
* Identify associations between objects.
* Identify attributes of objects and links.
* Organize and simplify object classes using inheritance.
* Verify that access paths exist for likely queries.
* Iterate and refine the model.
* Group classes into modules.

b. Dynamic Model

The dynamic model describes those aspects of a system concerned with time and the sequence of operations, such as events that mark changes, sequences of events, states that define the context for events, and the organization of events and states [Ref. 5:p. 18]. This model is important for interactive systems. The model captures control, that aspect of a system that describes the sequences of operations that occur, regardless of what the operations do, what they operate on, or how they are implemented.

The dynamic model is graphically represented with state diagrams. Each state diagram shows the state and event sequences permitted for one class of objects. The steps necessary to construct a dynamic model are as follows [Ref. 5:p. 261]:

* Prepare scenarios of typical interaction sequences.

* Identify events between objects and prepare an event trace for each scenario.
* Prepare an event flow diagram for the system.
* Develop a state diagram for each class that has important dynamic behavior.
* Check for consistency and completeness of events shared among state diagrams.

c. Functional Model

The functional model describes those aspects of a system that are concerned with value transformations. These include functions, mappings, constraints, and functional dependencies. The model captures what a system does without regard for how or when it is done.

The model is represented with data flow diagrams, which shows dependencies between values and the computation of output values from input values and functions. The processes on a data flow diagram correspond to activities or actions in the state diagrams of the classes. The flows on a data flow diagram correspond to objects or attribute values in an object diagram. The steps necessary to construct a functional model are as follows [Ref. 5:p. 261]:

* Identify input and output values.
* Use data flow diagrams as needed to show functional dependencies.
* Describe what each function does.
* Identify constraints.
* Specify optimization criteria.

The analysis model should include information that is meaningful from a real-world perspective and should present the external view of the system. Additionally, it should define the true requirements for a system. Once a problem has been analyzed, the solution must be designed.

2. System Design

System design is the high level strategy for solving the problem and building the system. It involves making decisions about the organization of the system into subsystems, the allocation of subsystems to hardware and software components, and major conceptual and policy decisions that form the framework for detailed design [Ref. 5:p. 198]. The overall structure and style of the system are decided.

The following steps are recommended for system design [Ref. 5:p. 199]:

* Organize the system into subsystems.
* Identify concurrency inherent in the problem.
* Allocate subsystems to processors and tasks.
* Choose an approach for management of data stores.
* Handle access to global resources.
* Choose the implementation of control in software.
* Handle boundary conditions.
* Set trade-off priorities.

The final form of the high-level structure of the system (determined during this phase) is called the system architecture. System architecture's can consist of many frameworks. These include functional transformations (such as batch processing or continuous transformations), time-dependent systems (such as interactive interface or real-time systems), and database systems. Most application systems are usually a combination of several forms.

3. Object Design

The analysis phase determines what the implementation must do. The system design phase determines the plan of attack. The object design phase determines the full definitions of the classes and associations used in the implementation, as well as the

interfaces and algorithms of the methods used to implement operations [Ref. 5:p. 227]. Object design is analogous to the preliminary design phase of the traditional software development life cycle.

During object design the designer carries out the strategy chosen during system design and pulls out the details. There is a shift in emphasis from application domain concepts toward computer concepts. The operations identified during analysis must be expressed as algorithms, with complex operations decomposed into simpler operations. The classes, attributes, and associations from analysis must be implemented as specific data structures [Ref. 5:p. 227]. New object classes may have to be introduced in order to store intermediate results during program execution.

The following steps are recommended for object design [Ref. 5:p. 228]:

* Combine the three models to obtain operations on classes.
* Design algorithms to implement operations.
* Optimize access paths to data.
* Implement control for external interactions.
* Adjust class structure to increase inheritance.
* Design associations.
* Determine object representations.
* Package classes and associations into modules.

It is imperative that all design decisions be documented when and where they are made. This documentation is often the best way of transmitting the design to others and recording it for reference later.

C. OMT AND OTHER OBJECT-ORIENTED APPROACHES

OMT is not the only object-oriented approach to software design that exists. The OMT methodology builds on earlier object-oriented work. Some of this earlier work was performed by several recognized leaders in the object-oriented software design field, to include Booch, Meyer, Shlaer and Mellor, and Coad and Yourdon. A brief overview of their respective works will be given.

Booch describes the rudiments of object-oriented software development He explains that object-oriented development is fundamentally different from traditional functional approaches to design [Ref. 7]. In later work, he extends Ada oriented work to the entire object-oriented design area. His methodology includes a variety of models that address the object, dynamic, and functional aspects of a software system [Ref. 8]. He places less emphasis on analysis and more on design.

Meyer does not present a methodology per se, however, he does provide many good tips on object-oriented design. He does not deal with conceptual modeling or analysis [Ref. 5].

Shlaer and Mellor describe a complete methodology for object-oriented analysis and design. They also break analysis down into three phases: static modeling of objects, dynamic modeling of states and events, and functional modeling [Ref. 9]. They caution that their methodology is an approach to analysis, and that final design might be different.

Coad and Yourdon's approach is similar to that of OMT's, with less emphasis on design [Ref. 10].

In the next chapter, OMT concepts and techniques will be applied to a specific Marine Corps software system.

IV. APPLICABILITY OF OBJECT-ORIENTED METHODOLOGY TO MARINE CORPS APPLICATIONS

In this chapter, a typical Marine Corps application (a Personnel Management System at the Company level) will be designed using both the current methodology and the proposed OMT model. This will demonstrate that OMT is applicable to Marine Corps non-tactical software design projects.

A. APPLICATION BACKGROUND

The application to be developed is not an existing system, but one that closely resembles several existing PC based applications throughout the Marine Corps. The application is a personnel management system, to be used by administrative personnel at the Company level. For lack of a better name, let's call it COPS (COmpany Personnel System).

COPS will maintain all pertinent information on Company personnel. It will perform report generation, compute promotion and physical fitness test scores, compile duty rosters, and allow for updates, deletions, and additions to the system.

COPS will be designed below using current Marine Corps Methodology.

B. COPS DESIGN USING SDM

This design will address only those aspects that pertain to software development. Specifically, it will address the Requirements Statement, Functional Requirements Definition, General Design Specification, and Detailed Design Specification.

1. Requirements Statement

a. General

Initial Problem Statement:

The purpose of the personnel management system is to help Commanding Officers/administrative personnel at the company level manage pertinent information on all personnel within their command. This will include report generation, updates, additions, deletions, retrieval of information, and computation of scores.

b. Current System

Currently, no automated personnel system exists at the company level. All data is maintained manually.

c. Required Capabilities

The new system should have the capabilities to perform the following tasks:

* update of information.
* addition of personnel to system.
* deletion of personnel from system.
* generate and compile necessary reports.
* generate necessary rosters.
* retrieve all necessary information on company personnel.
* compute promotion scores.
* compute Physical Fitness Test (PFT) scores.

The system will not interact with other systems. It should be a Local Area Network (server) based system. It should be a multi-user system. There is a requirement for the system to be backed up at least weekly.

2. Functional Requirements Definition

a. General

The COmpany Personnel System (COPS) is a multi-user, Local Area Network based application, designed to support Commanding Officers/administrative personnel in the management of information on all personnel in the command. This application is intended for use at the Company level. It should be applicable to most Company's throughout the Marine Corps.

b. Structural Specification

(1) Functional Requirements. The COPS system will perform the following functions:

* Provide a roster of all company personnel. The system should allow for rosters to be printed alphabetically, by platoon, by Military Occupational Specialty (MOS), by rank, or by any combination of the above.
* Provide for the computation of promotion (cutting) scores for all enlisted personnel within the company.
* Provide reports of promotion scores.
* Provide for the computation of PFT scores for all company personnel. This includes a report by numerical score, as well as by class.
* Provide means to update all fields within the system.
* Provide means to delete personnel from the system.
* Provide means to add personnel to the system.
* Provide means to add/delete fields to the system.
* Determine and print out duty lists (OOD, SDNCO, DNCO, etc.).
* Keep track of Company training.
* Keep track of required personal training requirements.

(2) Non-functional Requirements.

* The system must be a multi-user system.
* The application will reside on a LAN server.
* The response to a command/request should be no longer than 3 seconds.
* The system should be designed in accordance with DOD/Marine Corps standards.
* The system should be completed within 12 weeks.

(3) Context Diagram. The overall system is represented by the Context Diagram in Figure 5, below.

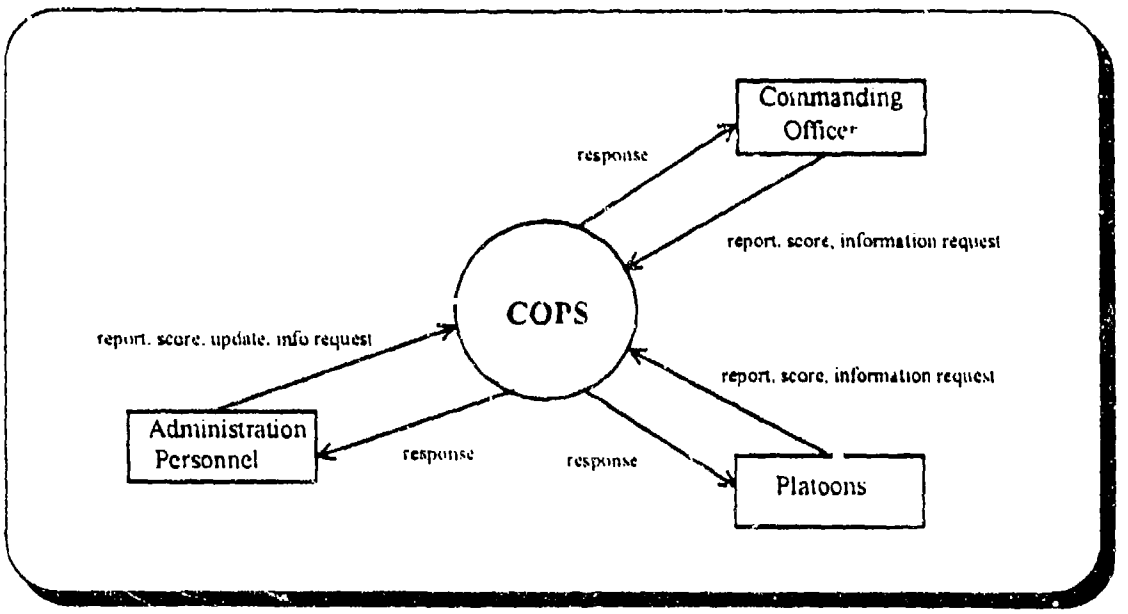

Figure 5 COPS Context Diagram

3. General Design Specification

a. General

The objective of the GDS is to provide a high-level view of the major functions within the COPS system. Additionally, it will describe the major interfaces between those functions. This will be accomplished via a Data Flow Diagrams (DFD), a Data Dictionary, and an Entity-Relationship Diagram (ERD).

b. DFD of COPS System

Figure 6 displays the DFD for the COPS system:

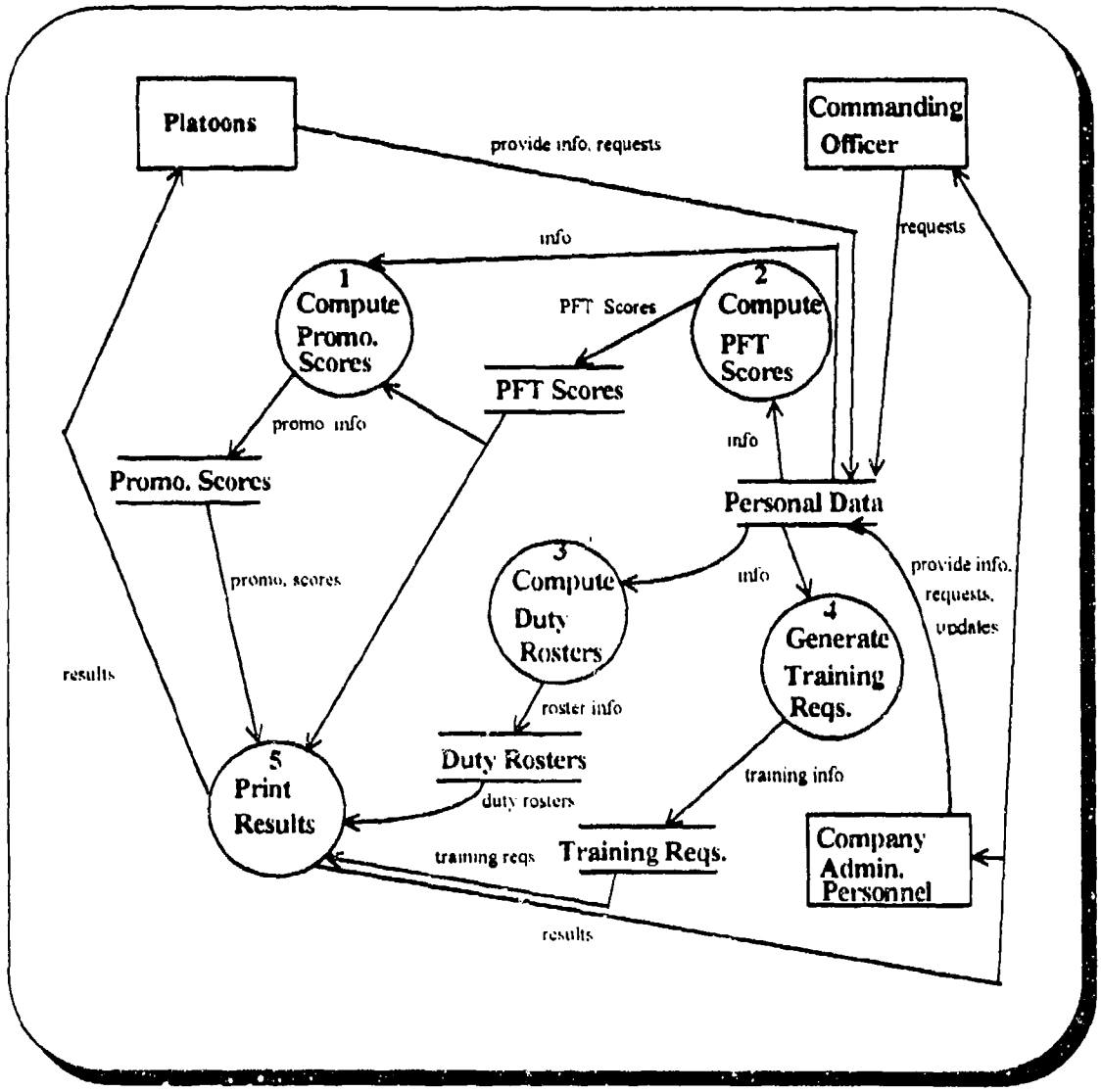

Figure 6 DFD for COPS

c. Data Dictionary fo · COPS

The following data makes up the Data Dictionary for COPS. The elements apply to the DFD as well as the ERD (to follow the Data Dictionary).

* address = street + city + state + zip

```
* age = [(int-num)]2
* ASVAB = [(int-num)]3
          **Armed Services Vocational Aptitude Battery Test Score**
* birth_date = yr + mo + day
* city = {legal-char}
* college = [(int-num)]2
          **number of college courses taken**
* company = [A|B|C|HQSVC]
* CRT = [Y|N]
        **combat readiness training, is it completed?**
* daily_trng_sch = {legal-char}
                   **daily training schedule**
* dependents = [(int-num)]1
               **number of dep svc member has**
* DNCO = DNCO
          **Duty NCO**
* Drug_Al = [Y|N]
          **drug and alcohol training, is it needed?**
* Duty Roster = **duty roster store**
* duty_qual_for = [mess_duty|DNCO|SDNCO|OOD]
* EST = [pass|fail]
        **Essential Subjects Test results**
* eye_color = {legal-char}
* first_name = {legal-char}
* GCT = [(int-num)]3
        **Armed Services I.Q. Test**
* hair_color = {legal-char}
* height = *units: inches, range: 46-84*
* int-num = [(0-int'last)]
* last_duty_date = yr + mo + day
                   **last time svc member stood duty**
* last_name = {legal-char}
* legal-char = [A-Z|a-z|0-9|'|-| |]
* MCI = [(int-num)]2
        **number of correspondence courses completed by svc member**
* mess_duty = Mess Duty
              **duty in the mess hall**
* MI = {legal-char}
      **middle init**
* MOS = [(int-num)]4
        **Military Occupational Specialty**
* name = last_name + first_name + MI
* OOD = OOD
```

```
        **Officer of the Day**
*  p_mark = [exp|ss|mm]
          **pistol marksmanship class**
*  p_score = [(int-num)]3
          **pistol score**
*  PFT Scores = **PFT scores store**
*  PFT_class = [1st|2nd|3rd|fail]
*  PFT_score = [(int-num)]3
*  phone_num = {legal-char}
*  pistol_trng =  [Y|N]
              **pistol training, is it completed?**
*  PLT =  [1st|2nd|3rd|wpns]
         **platoon**
*  promo_score = [(int-num)]4
                **promotion/cutting score**
* Promotion Scores = **promotion scores store**
*  pull_ups = [(int-num)]2
            **num of pull-ups**
*  pull_up_score = [(int-num)]3
*  r_mark = [exp|ss|mm]
          **rifle marksmanship**
*  r_score = [(int-num)]3
            **rifle score**
* rank = [Pvt|LCpl|Cpl|Sgt|SSgt|GySgt|MSgt|1stSgt|MGySgt|SgtMaj|2ndLt|1stLt|
        Capt|Maj|LtCol|Col]
*  rel_pref = {legal-char}
            **religious preference of svc member**
*  rifle_trng = [Y|N]
             **rifle training, is it completed?**
*  run_score = [(int-num)]3
*  run_time = *unit: minutes, seconds :range : 0-60*
             **time on run portion of PFT**
*  SDNCO = SDNCO
             **Staff Duty NCO (E-6 and above)**
*  sex = [M|F]
*  sit_ups = [(int-num)]2
*  sit_up_score = [(int-num)]3
*  SSN = {(int-num)3 - (int-num)2 - (int_num)4}
       **Social Security Number**
*  Training Req = **training requirements store**
*  weekly_trng_sch = {legal-char}
                   **weekly training schedule for company**
*  weight = *units:pounds; range: 1-400*
```

d. Entity-Relationship Diagram for COPS

The following ERD is used to display the stored layout of the COPS system at a high level of abstraction. For the sake of clarity, the attributes of each entity have been omitted from the diagram. Figure 7 displays the ERD.

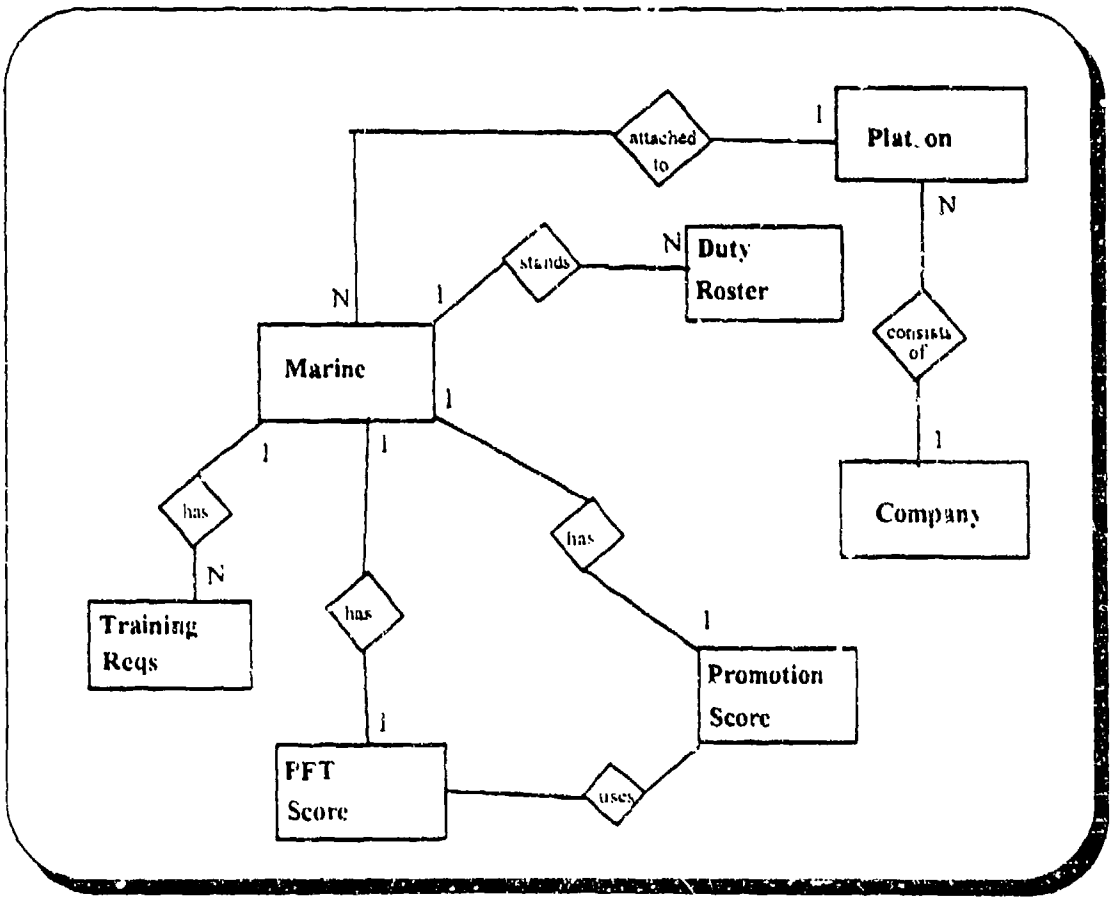

Figure 7 ERD for COPS

4. Detailed Design Specification for COPS

a. General

The objective of the DDS is to provide sufficient detail of each module defined in the DFD's so that these modules can be coded by a programmer. Additional DFD's as

well as module specifications will be designed and developed in this phase of the DDS. For the purpose of this thesis, only one module of the COPS system will be designed in detail.

b. Structural Specification

(1) DFD of Compute PFT Scores Module. The "Compute PFT scores" module of the COPS system will be broken down into detail. Figure 8 is a DFD of the module.

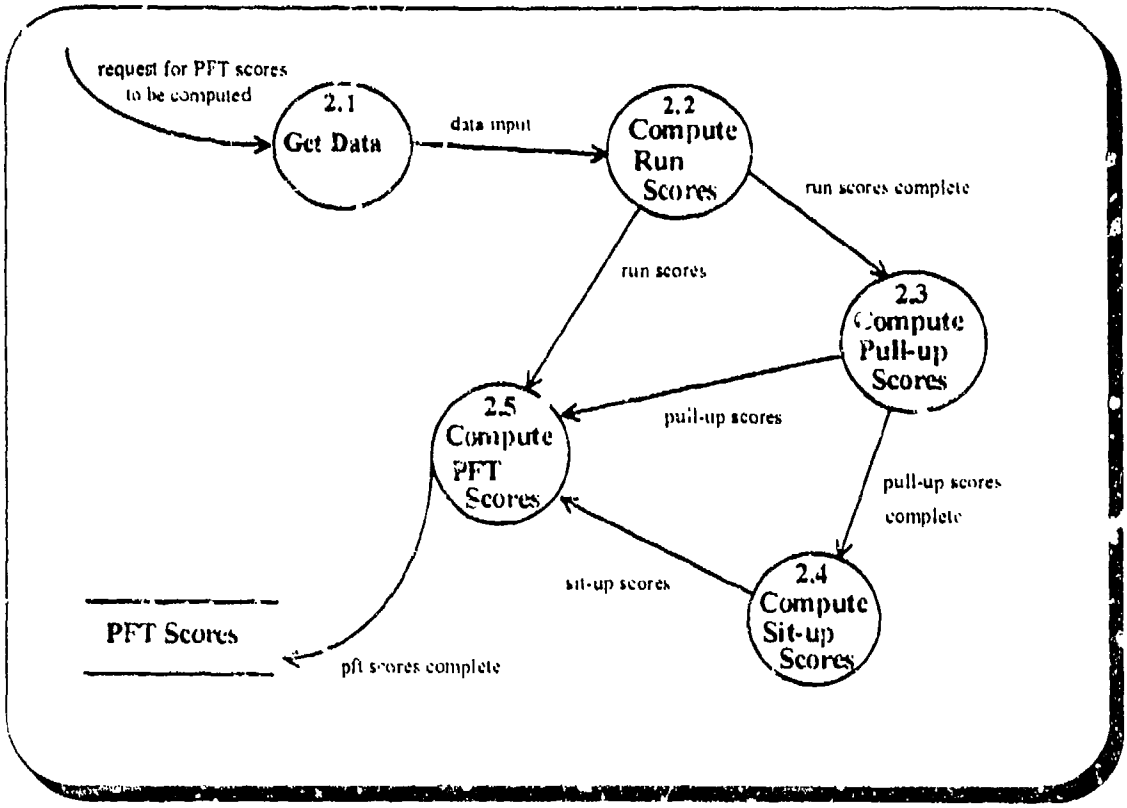

Figure 8 DFD for "Compute PFT Scores" Module

(2) Module Specifications. Module Specifications (MS) is a statement of the rules governing the transformation of input data into output data. Module Specifications

define the control logic for system execution. MS for the "Compute PFT Scores" module will be described below.

Compute PFT Scores: Figure 6: Module 2

Do While there are more Marines in the Company
 Get PFT data fields (age, ssn, run_time, sit_ups, pull_ups, hang_time, sex)

Compute Run Scores: Figure 8: Module 2.1

Do While there are more Marine run_time values
 Do Case
  Case sex = "m"
   use male_chart table
    if run_time.seconds >= 01 and <= 09 then round seconds value up to 10
    else if run_time.seconds >= 11 and <= 19 then round seconds value up to 20
    else if run_time.seconds >= 21 and <= 29 then round seconds value up to 30
    else if run_time.seconds >=31 and <= 39 then round seconds value up to 40
    else if run_time.seconds >=41 and <= 49 then round seconds value up to 50
    else if run_time.seconds >=51 and <= 59 then round seconds value up to 60
    end if
   Determine run_score based on comparison between run_time and male_chart value
   Write run_score to "Compute PFT Score" module
  Case sex = "f"
   use female_chart table
    if run_time.seconds >= 01 and <= 09 then round seconds value up to 10
    else if run_time.seconds >= 11 and <= 19 then round seconds value up to 20
    else if run_time.seconds >= 21 and <= 29 then round seconds value up to 30
    else if run_time.seconds >= 31 and <= 39 then round seconds value up to 40
    else if run_time.seconds >= 41 and <= 49 then round seconds value up to 50
    else if run_time.seconds >= 51 and <= 59 then round seconds value up to 00

```
      end if
     Determine run_score based on comparison between run_time and
      female_chart value
     Write run_score to "Compute PFT Scores" module
    Case Otherwise display "No entry value for "sex" field of Marine, try again"
    End Case
  End Do
```

Compute Pull Up Scores: Figure 8: Module 2.2

```
 Do While there are more pull_up/hang_time values
   Do Case
    Case sex = "m"
     get pull_ups and Multiply by 5 and assign value to pull_up_score
     Write pull_up_score to "Compute PFT Scores" module
    Case sex = "f"
     use female_chart table
     get hang_time value
      if hang_time <= 40 then hang_score is assigned hang_time value
     else determine hang_score based on comparison between hang_time and
     female_chart value
     end if
    Write hang_score to "Compute PFT Scores" module
   End Case
  End Do
```

Compute Sit Up Scores: Figure 8: Module 2.3

```
  Do While there are more sit_up values
   get sit_ups value
   Do Case
   Case sex = "m"
    use male_chart table
     if sit_ups <= 60 then sit_up_score is assigned sit_up value
    else determine sit_up_score based on comparison between sit_ups and
    male_chart table
    end if
   Write sit_up_score to "Compute PFT Scores" module
   Case sex = "f"
    use female_chart table
```

```
    Determine sit_up_score based on comparison between sit_ups and
     female_chart table
    Write sit_up_score to "Compute PFT Scores" module
   End Case
  End Do
```

Compute PFT Scores: Figure 8: Module 2.4

```
 Do While there are more PFT Scores
  if age >= 17 or <= 26 then use junior_marine_scoring table
  else if age >= 27 or <= 39 then use mid_level_marine_scoring table
  else age >= 40  then use senior_marine_scoring table
   end if
  Do Case
    Case sex = "m"
     Add run_score + pull_up_score + sit_up_score to get PFT_score
      Determine PFT_Class based on comparison between PFT_score and
      junior/ mid_level/ senior_marine_scoring table
     Write PFT_score and PFT_class to PFT Scores data store
    Case sex = "f"
     Add run_score + hang_score + sit_up_score to get PFT_score
     Determine PFT_Class based on comparison between PFT_score and
     junior,mid_level, senior_marine_scoring table
     Write PFT_score and PFT_class to PFT Scores data store
   End Case
  End Do
 End Do  --Main Loop
```

C. COPS DESIGN USING OMT

COPS will be utilized to display the three kinds of models under OMT; the Object, Dynamic, and Functional Models. These three models were described in detail in Chapter III. The problem statement and functional requirements for COPS (described in section B) remain the same.

I. Analysis Phase

a. Object Model for COPS

Figure 9 displays the COPS object model. For clarity, the attributes have been omitted from the diagram.

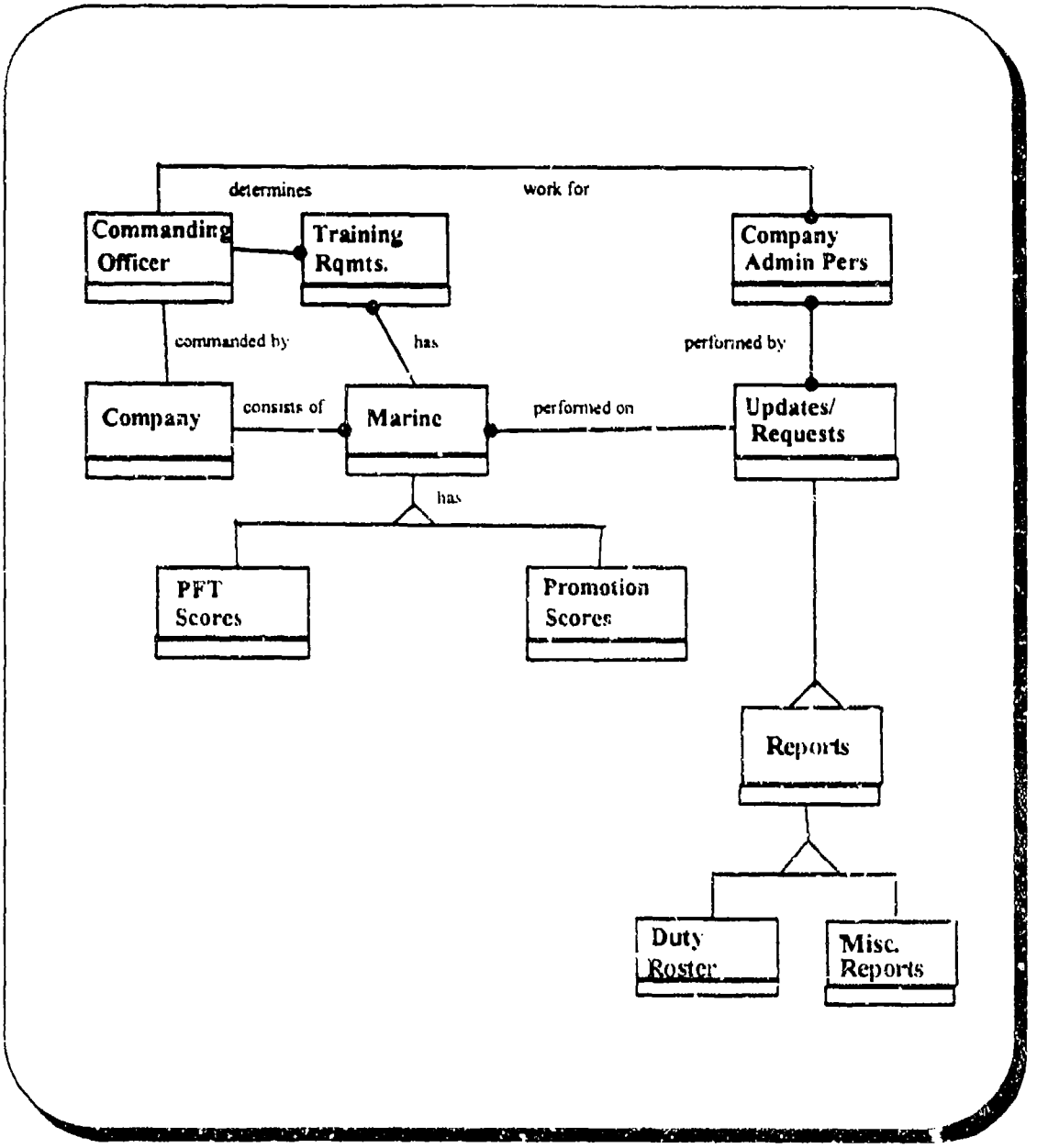

Figure 9 COPS Object Model

(1) Data Dictionary for Object Model. The data dictionary describes each object class defined within the object model.

* Commanding Officer - This object defines the commanding officer for the company.
* Company - This object defines the type of company in the COPS system.

* Company administration Personnel - This object defines the personnel authorized to access the COPS system. It contains names, ssn's, passwords, and ranks.
* Deletions - This class inherits all the methods and attributes of its parent object, Updates/Requests. This object allows for marines, attributes, training rqmts, etc. to be deleted from the system.
* Duty Roster - This object inherits all the methods and attributes from its parent object Reports. It contains information on the different types of duty, as well as personnel.
* Marine - This object contains all the information for all marines in the company.
* Misc. Reports - This object inherits all the methods and attributes from its parent object Reports. It allows for ad-hoc reports to be created based on requirements.
* PFT Scores - This class inherits all the methods and attributes of its parent object, Marine. This object contains pft scores for all marines in the COPS system. It is used by other objects as well.
* Promotion Scores - This class inherits all the methods and attributes of its parent object, Marine. This object contains promotion scores for all marines in the COPS system. It is used by other objects as well.
* Reports - This object contains the reports formats. It also allows the user to select which reports he/she wants to generate.

b. Dynamic Model

The dynamic model is very important for interactive systems. The COPS system can be described as a data repository system, or a database system, that is not highly interactive with the user. Therefore, the dynamic model will be limited in its detail. Figure 10 displays the Event Flow Diagram (EFD) for the COPS system.

(1) Event Flow Diagram for COPS. The EFD summarizes events between classes, without regard for sequence.

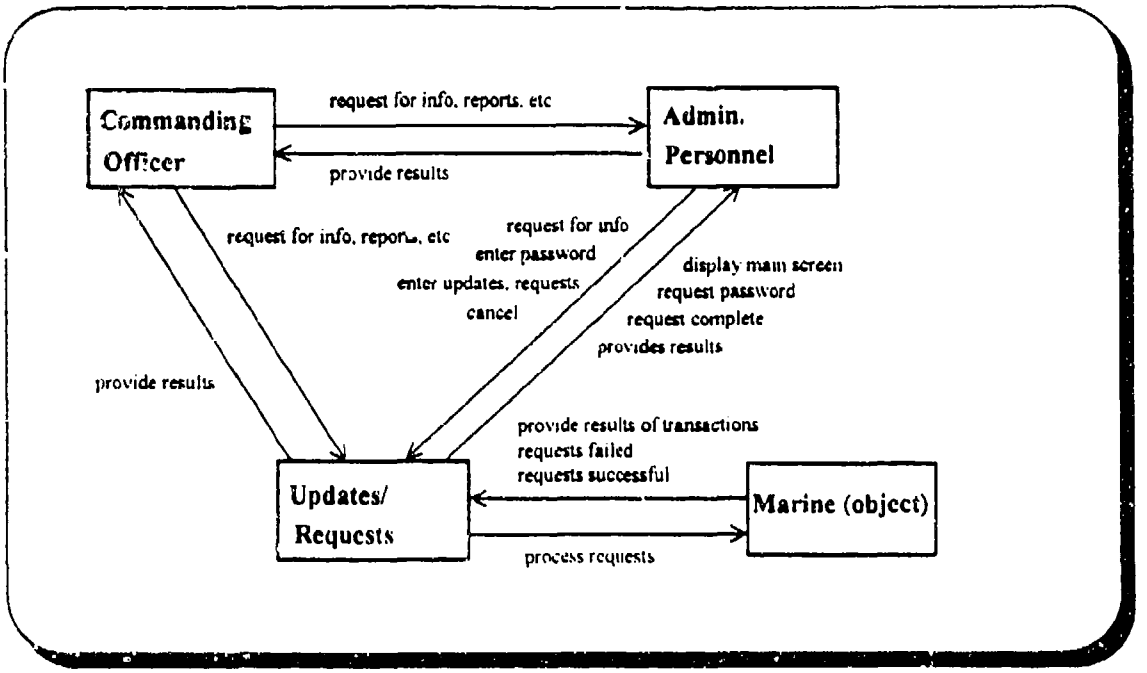

Figure 10 Entity Flow Diagram for COPS

(2) State Diagram for COPS. A state diagram for each object class is the next step in the process. The state diagram captures all the events the object receives and sends. For the purpose of this thesis, only the object class "PFT Scores" will be designed. Figure 11 displays the state diagram for "PFT Scores."

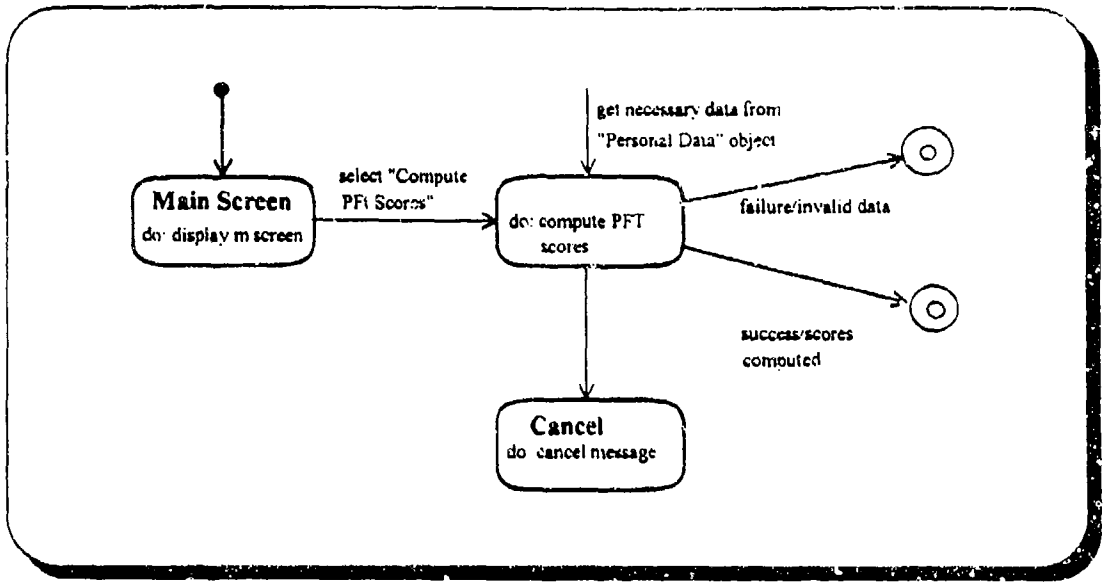

Figure 11 State Diagram for "PFT Scores"

c. Functional Model

The functional model shows how values are computed, how they depend on which other values and the functions that relate to them. DFD's are useful for showing functional dependencies. A DFD is designed for the entire system, followed by a DFD for each object class. Figure 6 from the GDS can be used here in the OMT model, since it describes the top-level functionality of the COPS system. There is no need to display it her, since it already has been done. Additionally, Figure 8 from the GDS can be used to display the functionality of the "PFT Scores" object.

2. System Design

In the system design phase of OMT, the overall structure and style of the system are decided. The first step in the process is to break the system into subsystems. Each

subsystem encompasses aspects of the system that share some common property, in our case similar functionality. The subsystems for the COPS system are as follows:

* Users - this subsystem includes the objects Commanding Officer and Company Admin. Personnel.
* Marine Data - this subsystem includes the objects Marine, PFT Scores, and Promotion Scores.
* Updates and Reports - this subsystem includes the objects Updates/Requests, Reports, Duty Roster, and Misc. Reports.

The next step in the process is to determine an approach for management of data stores. It comes down to whether or not the system requires the use of a database, and if so, what type to use. In our case, COPS is a data intensive system, thus requiring the use of a database.

The overall system architecture is determined next. What we are trying to do in this step is match application behavior with architectural framework. In our case, the COPS system can be classified as a transaction/database management system, thus requiring a transaction management system architecture. The main function of this system is to store, access, and perform computations on information.

3. Object Design

During object design the designer carries out the strategy chosen during system design and pulls out the details. During this phase, the actual algorithms for each operation are designed. The operations are determined by combining the three models of OMT's analysis phase. In our case, the "Compute PFT Scores" operation has been determined. The algorithm for this operation has already been defined in the DDS, and will not be reproduced here.

In the next chapter, a comparison between the SDM and OMT methodologies will be performed.

V. COMPARISON BETWEEN SDM AND OMT METHODOLOGIES

The purpose of this chapter is to clearly identify the major differences and similarities between SDM and OMT. It is important to remember that SDM is based on the traditional structured analysis/structured design approach to software development and design.

A. GENERAL OBSERVATIONS

SDM and OMT, although different in their approach to software development and design, have much in common. Both models use similar constructs and support the three orthogonal views of a system. The difference between SDM and OMT is primarily a matter of style and emphasis. In the SDM approach, the functional model dominates, followed by the dynamic model, and then the object model. In contrast, OMT regards the object model as most important, followed by the dynamic model, and finally the functional model. Several areas will be addressed in the comparison between SDM and OMT.

1. Organization

SDM organizes a system around procedures, while OMT organizes a system around real-world objects, or conceptual objects that exist in the user's view of the world.

Most changes in system requirements pertain to function rather than to objects, so change can be disastrous to procedure-based design . By contrast, changes in function are readily accommodated in an object-oriented design by adding or changing operations, while leaving the basic object structure unchanged.

An example that displays how SDM organizes around procedures can be found in Figure 6 (DFD for COPS). The DFD is done at the beginning of the design process, and it breaks down the system into separate "modules" or "procedures." By contrast, OMT begins the design process with the Object Model (shown in Figure 9). This model organizes the system around objects.

2. Extendibility

An SDM design has a clearly defined system boundary, across which the software procedures must communicate with the real world [Ref. 5:p. 268]. The overall structure of an SDM design is derived from the system boundary, so it can be difficult to extend an SDM design to a new boundary. By contrast, it is much easier to extend an object-oriented design to a new boundary. This is done by merely adding objects and relationships near the boundary to represent objects that existed previously only in the outside world. OMT is more resilient to requirement changes and therefore more extensible.

The system boundary for SDM can be found in Figure 5. The COPS system must interact with three entities, Commanding Officer, Platoons, and Administration Personnel. These entities are displayed in Figure 5 because they have to interact with

COPS. If we wanted to add an entity, it would be very difficult. First, we would have to create this entity from scratch (specific code for the entity). Second, we would have to define its functionality and somehow incorporate this into existing logic and functionality.

The system boundary for OMT can be found in Figure 10. There are four objects displayed, Commanding Officer, Administration Personnel, Updates/Requests, and Marine. Each of these objects exist in the COPS system. As with SDM, they must also interact inside COPS, however, in contrast to SDM, they are actually objects defined in the system.

Let's say there are requirement changes in the COPS system. Specifically, the user wants to change the way some information is represented in the system: he wants to add several attributes to "Marine." Additionally, the user wants to add another operation to COPS; compute rifle range scores from raw data. With these additional changes, the following will have to happen in the system designed using SDM:

* In order to add the attributes, the "Marine" entity in the database being used with COPS will have to be modified. These changes could force changes elsewhere in the system, sort of a rippling effect.
* From a designers view, a "Compute Rifle Range Scores" module will have to be added to the DFD in Figure 6. This module will have to be inserted at the proper location, since SDM design is based on functionality.
* Specific program code will have to be developed in order to be able to compute the rifle range scores, and then pass the results on to the necessary operations.

In contrast, the following limited changes will have to happen in the system designed using OMT:

* The required attributes will be added to the "Marine" object, as well as the additional operations.

* A child object "Rifle Range Scores" will be created under its parent "Marine."
* Program code will have to be added to perform the required operation.

3. Understandability

In OMT, the direct analogy between objects in the design and objects in the problem domain results in systems that are easier to understand. This understandability makes the design more intuitive and simplifies traceability between system requirements and program code. It also makes the design more coherent to persons who are not a part of the original design team.

The Object Model (Figure 9) displays the objects to be used in the COPS system. The Object Model was developed over several iterations. During each iteration, objects were scrutinized for redundancy and relevancy. Once a correct Object Model is defined, the objects contained within are used throughout the design and implementation. Functions are performed on these objects, while the object itself is not altered or changed.

In contrast, SDM defines the functions to be performed in the system (Figure 6) and then defines how the data is to be organized in the database (Figure 7). The designers of the DFD and ERD may use different names in their respective diagrams to refer to the same set of data or functions. This can lead to confusion among the developers. In OMT, the system is designed around objects, and once the objects are designed and named, they are used consistently throughout implementation.

4. Inheritance

In 3DM, the system is designed around its functionality, which usually means that most program code is specific and not easily used elsewhere. By contrast, systems designed using OMT use objects as the basis for their organization. This allows for the basic object to be reused over and over again. This also allows for objects to be easily designed for the system. This ease of design increases reusability of components and objects from one project to the next.

Figure 9 displays the Object Model for COPS. The "Marine" object is considered a parent object while "PFT Scores" and "Promotion Scores" are considered its children. What this means is that all functions (methods) and attributes contained in "Marine" are automatically inherited by its children, thus saving time and effort with respect to program code.

What should be emphasized here is that inheritance can be used very effectively in Marine Corps applications because of its hierarchical nature in data organization. The most important (or common) data is maintained within the parent object, and data that is specific to an object is maintained within that child object. For example, the single most important attribute (Social Security Number) is maintained in the "Marine" object. The children of this object ("PFT Scores" and "Promotion Scores") inherit this attribute from their parent, while maintaining attributes that are specific to their operations. The most important data resides at the top, thus allowing for data to be maintained in a hierarchical nature.

5. Database Integration

A system designed around functionality is inherently awkward at dealing with databases because it is difficult to merge programming code organized about functions with a database organized about data. This is not always the case, but it is generally accepted that merging the two requires time and patience. By contrast, an object-oriented approach does a better job at integrating databases with program code. This can be attributed to the use of one uniform paradigm, the object. The object can model both database and programming structure. For instance, if a database were designed for COPS (COPS designed with OMT), the Entity Relationship Diagram (used for database design) would have the same entities as objects defined in the Object Model (Figure 9). In OMT, the objects defined in Figure 9 will be used throughout design and implementation. By contrast, SDM uses DFD's to display functionality (Figure 6) while using an ERD (Figure 7) to define database design. The two may or may not use the same names for both diagrams. This can lead to confusion, as well as to problems with database integration.

6. Maintenance

What does the term "maintenance" mean with respect to software? The term addresses two activities: modifications and debugging. Modifications can be defined as changes in the external world (of the user) that require changes to the computer system. Additionally, modifications can be defined as debugging efforts; removing errors that should never have been there in the first place.

Systems developed with OMT tend to be easier to modify (change) because they are organized around objects instead of functionality. If the designer wants to make changes to functionality, all he has to do is concentrate on code that pertains to functionality, while leaving code that pertains to objects alone. If the designer wants to change an object, all he has to do is go to that object and make changes, ignoring code that pertains to functionality.

By contrast, changes to systems developed with SDM tend to be more complex, time consuming, and costly because any changes to program code can effect many other functions within the system.

Let's assume that the system has been developed and is fully functional. Let's further assume that the user wants to add some functionality to the system. Specifically, they want the system to compute Essential Subjects Test (EST) scores for all enlisted Marines in the Company. In OMT, an object would be created called "EST Scores" that would be a child object of the parent object "Marine." It would inherit all existing methods and attributes from "Marine", thus saving time and effort. The specific functions to be performed on the object would have to be defined and coded.

In SDM, another module would have to be created for this process. The designer would be starting from scratch, and would have to concern himself with fitting this new module into a system designed around functionality, where slight changes in one area could have dramatic effects in others.

Debugging efforts are never easy, whether the system was developed using SDM or OMT. What comes into play here is the amount of experience and knowledge possessed by programming personnel.

B. CHOOSING BETWEEN SDM AND OMT

This section will graphically display when it is advantageous to use SDM or OMT for system development. Figure 12 displays a table which serves as a quick reference. The table references small, medium, and large applications. Small applications (for the purpose of this thesis) are defined as applications which have less than or equal to 3000 lines of code. Medium sized applications are defined as having greater than 3000 and less than or equal to 20,000 lines of code. Large applications have greater than 20,000 lines of code.

	Small Application	Medium Application	Large Application
ease of extendability	SDM OMT	OMT	OMT
ease of understandability	SDM OMT	SDM OMT	OMT
ease of inheritance (use)	OMT	OMT	OMT
ease of DataBase integration	SDM OMT	SDM OMT	OMT
ease of maintenance	SDM OMT	SDM OMT	SDM OMT

Figure 12 Comparison of SDM and OMT Methodologies

The next chapter will address the conclusions and make specific recommendations about the required changes to the status quo if OMT is to be incorporated as a software development methodology for Marine Corps applications

VI. CONCLUSIONS AND RECOMMENDATIONS

Object-Oriented Methodologies, such as OMT, can be used by the Marine Corps for its "non-tactical" software development projects in the near future. The current methodology, SDM, is still valid and should be used where applicable. As a matter of fact, OMT utilizes certain portions of the traditional software design methodology.

A. CONCLUSIONS

This thesis addressed three basic questions: (1) What is Object-Oriented software design and why is it good for the Marine Corps, (2) How is it different than what we are doing, and (3) What should we do and when should we do it?

Question one was addressed in Chapter III. A specific Object-oriented methodology (OMT) was described. Object-oriented software development was defined as a new way of thinking about software based on abstractions that exist in the real world. The essence of this development is the identification and organization of application-domain concepts, rather than their final representation in a programming language, object-oriented or not. The greatest benefit of an object-oriented approach is that it helps specifiers, developers, and customers express abstract concepts clearly and communicate them to each other. It can serve as a medium for specification, analysis, documentation, and interfacing, as well as for programming.

Object-oriented software development is good for the Marine Corps because it possesses many benefits. First, the data are organized into discrete, distinguishable entities called objects, each possessing its own "identity." Second, objects with the same attributes and functions are grouped together into a single class. This is known as "classification," a form of abstraction. Third, it allows for the same operation to be called by many different classes where this operation may behave differently in each class. This is known as "polymorphism." Fourth, it allows for the sharing of attributes and functions among classes based on a hierarchical relationship. This concept is known as "inheritance." Fifth, it allows for "encapsulation," also known as information hiding. This consists of separating the external aspects of an object from the internal implementation details.

In order to address question two, the design methodology currently used by the Marine Corps (SDM) was defined in Chapter II. SDM is based on DOD and DON standards and procedures known as "Life Cycle Management." SDM is based on traditional software development activities commonly known as the software life-cycle. The life-cycle approach involves the following activities: requirements analysis, functional requirements definition, general design specification, detailed design specification, operation, and maintenance.

In order to show how OMT and SDM differ, a hypothetical system (COPS, which is based on several existing systems) was developed and designed in Chapter IV. In order to point out where the methodologies differ, a comparison and contrast was performed

between the two in Chapter V. The basic difference between the two is primarily a matter of style and emphasis. In SDM, the functional model dominates, followed by the dynamic model, and finally the object model. OMT regards the object model as most important, followed by the dynamic model, and finally the functional model. Additionally, there were six categories compared between the two: organization, extendibility, understandability, inheritance, database integration, and maintenance.

SDM organizes a system around functionality, while OMT organizes a system around real-world objects, or conceptual objects that exist in the user's view of the world. If system requirements change, these changes will be easier to incorporate into a system designed with OMT than with one designed with SDM.

OMT is more resilient to requirement changes and therefore more extensible. In order to extend a system designed with OMT the designer merely adds objects. With SDM, the overall structure must be changed in order to extend the system.

With respect to understandability, OMT utilizes a direct analogy between objects in the design and objects in the problem domain. This makes it easier to understand and follow throughout. By contrast, SDM defines functions to be performed and then defines how the data is to be organized.

With respect to inheritance, OMT allows for an object to be used over and over again, thus allowing for extensive use of inheritance. In contrast, a system designed with SDM is designed around functionality, which means that most program code is specific and not easily used (inherited) elsewhere.

OMT allows for easier integration between program code and databases because of its use of one uniform paradigm, the object. SDM organizes program code around functionality, while databases are organized around data. This makes it more difficult to integrate the two.

There are two components of maintenance: modifications and debugging. Systems designed with OMT tend to be easier to modify, while systems designed with SDM tend to be complex, time consuming, and costly. This is due to the manner in which the systems are organized. Debugging is never easy, no matter which approach is taken.

Question three will be addressed in the next section.

B. RECOMMENDATIONS

This section will address question three: What should the Marine Corps do and when should they do it? The Marine Corps should not (and cannot) discard SDM. SDM is a requirement for software development projects. What can be done is modification to the contents and documentation requirements of SDM. Specifically, the Functional Requirements Definition, General Design Specification, and Detailed Design Specification sections will have to be changed. These sections will have to be replaced by OMT's Analysis, System Design, and Object Design phases respectively. The documentation requirements would shift from the status quo to those outlined and required by OMT's three phases mentioned above.

In order to determine when the Marine Corps should incorporate OMT into SDM, it must first be evaluated if OMT is applicable to Marine Corps initiatives. In Chapter IV, a system (COPS) was developed using both methodologies. It would not be valid to say that OMT is applicable to all Marine Corps software projects, however, it can safely be said that OMT would be applicable to most personnel management, logistical, inventory control, and database systems designed for "non-tactical" use. This can be said since object-oriented development is fundamentally a new way of thinking and not a programming technique. Therefore, it is not restricted to its use with only object-oriented languages (C++, SmallTalk, etc.). Even as a programming tool, it can have various targets, including conventional programming languages and databases as well as object-oriented languages [Ref. 5:p. 5].

When should the Marine Corps fully adopt OMT? The Marine Corps should start the process immediately. They should familiarize their software engineers with OMT by allowing them to receive formal training, and then have them design several systems using OMT. Once feedback is received about OMT's applicability and potential benefits, then a timeline should be developed for full implementation of OMT into the life-cycle process.

LIST OF REFERENCES

[1] U.S. Marine Corps, Computer Sciences School DOB 0702, *Introduction to the Life Cycle and System Development Methodology for Information Systems Projects*, June 1986.

[2] U.S. Marine Corps, Information Resources Management (IRM) 5231-01, *System Development Methodology Overview*, 16 June 1987.

[3] Ince, D., *Software Engineering: The Decade of Change*, p. 2, Peter Peregrinus Ltd., 1986.

[4] Berzins, V. A., and Luqi, *Software Engineering with Abstractions*, p. 8, Addison-Wesley Publishing Company, 1991.

[5] Rumbaugh, James., and others, *Object-Oriented Modeling and Design*, p. 4, Prentice-Hall, Inc., 1991.

[6] Meyer, Bertrand., *Object-Oriented Software Construction*, p. 62, Prentice-Hall International Ltd., 1988.

[7] Booch, Grady., "Object-Oriented Development," *IEEE Transactions on Software Engineering*, 12, pp. 211-221, 2 February 1986.

[8] Booch, Grady., *Object-Oriented Design*, p. 57, Benjamin/Cummings, 1991.

[9] Mellor, Stephen J., and Shlaer, Sally., *Object-Oriented Systems Analysis: Modeling the World in Data*, p. 37, Yourdon Press, 1988.

[10] Yourdon, Edward., *Modern Structured Analysis*, p. 32, Prentice-Hall, Inc., 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Director, Training and Education MCCDC, Code C46 1019 Elliot Road Quantico, Virginia 22134-5027	1
4.	Commandant of the Marine Corps Code MISB Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
5.	Computer Technology, Code 32 Naval Postgraduate School Monterey, California 93943-5002	1
6.	C. Thomas Wu, Code 32 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5002	1
7.	Capt. Robert F. Padilla, Jr. 22 Jason Lane Stafford, Virginia 22554	1